公交网络路径规划问题中的一种高效索引方法*

马 慧[1]，汤 庸[2†]，梁瑞仕[1]

(1. 电子科技大学中山学院 计算机学院，广东 中山 528400; 2. 华南师范大学 计算机学院，广州 510631)

摘 要：TTL 是在公交网络中求解最早到达路径、最晚出发路径和最短耗时路径的一种高效索引。TTL 采用 Time-dependent Dijkstra 为核心算法构建索引，存在两个不足：大量的昂贵的出堆操作拖慢了建立索引的效率以及所求得的路径具有较多的换乘次数。针对这两个不足，提出了一种基于旅程的索引 TAIL。TAIL 预先生成部分路径，在查询阶段通过匹配部分路径得到最优解，避免在原图上做查询，提高效率。TAIL 并不是基于图结构，而是以旅程为单位存储公交数据。在生成路径时，首先扫描路过起点的旅程，找到从起点直达的站点；然后扫描从直达站点出发的旅程，找到一次换乘可达的站点；如是这般，从可达站点出发扫描旅程，发现更多的可达站点。为了在早期找到最早到达路径，从而减少旅程的扫描量，TAIL 并没有严格按照换乘次数的顺序扩展站点。这种方法避免了昂贵的堆操作，也保留了旅程的完整性。在真实数据集上测试表明，与 TTL 相比，TAIL 有较短的建立索引的时间，生成的路径的换乘次数也较少。

关键词：最短路径；公交网络；路径规划；索引；时间表；换乘次数

中图分类号：TPT311.13 doi: 10.3969/j.issn.1001-3695.2018.02.0088

Efficient index for route planning in public transportation networks

Ma Hui[1], Tang Yong[2†], Liang Ruishi[1]

(1. *Computer School*, *Zhongshan College of University of Electronic Science & Technology of China*, *Zhongshan Guangdong* 528400, *China*; 2. *Computer School*, *South Normal University*, *Guangzhou* 510631, *China*)

Abstract: TTL is a highly efficient indexing structure for finding an earliest arrival path, or a latest departure path, or a shortest duration path in public transportation networks. TTL uses Time-dependent Dijkstra's algorithm as its core algorithm to build index, and is therefore, results in two deficiencies. Firstly, it needs relatively expensive priority queue operations. Secondly, it would generate paths with more transfers. This paper proposes a new indexing structure, TAIL, which uses a trip based method to build index. TTL pre-computes some canonical paths. A query could be answered by matching up the canonical paths, which avoids traversing the entire network. Instead of the graph structure, TAIL uses trip array as its input, and generates paths by scanning trips. Initially, TAIL scans trips starting from the source stop, from which TAIL obtains direct reachable stops. After that, TAIL scans trips starting from the direct reachable stops, from which TAIL obtains reachable stops within one transfer. Generally, TAIL discovers new reachable stops from scanning the trips starting at the already discovered reachable stops. In order to obtain the earliest arrival paths in the early stage, so as to reduce the number of trip scanning, TAIL does not scan the stops strictly in increasing order of their transfer times. In this way, TAIL avoids valuable priority queue operations, while preserves the entity of a trip. Experiments on real datasets shows that, compared to TTL, TAIL has lower index construction time and its generated path has fewer transfer times.

Key words: shortest path; public transportation network; route planning; index; time table; transfer times

0 引言

公交网络中的路径规划问题是近期研究的一个热点。给定起始站点 s 和目标站点 d，路径规划问题求解以下三种路径：a) 最早到达路径(earliest arrival path， EAP),在 π 时刻或之后从 s 出发，最早到达 d 的路径是哪条？b) 最晚出发路径(latest departure path，LDP),要求在 π' 时刻或之前到达 d，最晚离开 s 的路径是哪条？c) 最短耗时路径(shortest duration path，SDP),期望在 π 时刻或之后离开 s，在 π' 时刻或之前到达 d，耗时最短的路径是哪条？

收稿日期：2018-02-23；**修回日期：**2018-03-29 **基金项目：**国家自然科学基金(61772211)；广东省高等学校优秀青年教师项目(YQ2015241、YQ2015242)；广东省青年创新人才类项目（2015KQNCX206）中山市科技计划项目（2015B2307）

作者简介：马慧（1981-），女，广东中山人，副教授，博士，主要研究方向为数据库理论、图数据查询；汤庸（1964-），男（通信作者），教授，博导，博士，主要研究方向为社交网络、大数据应用、协同计算等（ytang4@qq.com）；梁瑞仕（1982-），男，副教授，博士，主要研究方向为人工智能、自动规划.

目前，关于公交网络中的路径规划问题已经有了很多的研究工作。早期的算法都是直接在图上做查询的，随着图的规模增大，查询效率不如人意。因此，学者们预先对图做预处理，通过计算部分路径信息生成索引，然后在索引上做查询，通过索引中的路径信息得到查询结果，避免在原图上做盲目搜索。在众多索引算法中，TTL[1]是近期提出的一种最新的索引。实验表明，在拥有大约五万个顶点、四百万条边的图上，TTL 的平均查询时间在 30 μs 以内，比已有的索引算法更高效。TTL 采用图结构表示公交数据，用公交站点作为图的顶点，如果在时刻 π_{dept} 有一趟交通工具从站点 u 出发，在 π_{arr} 时刻到达 v，中途不停靠，则在图中添加一条从 u 到 v 的边，在边上标记时刻〈π_{dept}, π_{arr}〉。TTL 引入路网下最短路径的高效索引 2-Hub-Labelling[2]的思想对图建立索引，索引构建基于图的求解最短路径的经典算法—Time-dependent Dijkstra 算法[3]查找路径。但是，Time-dependent Dijkstra 算法有两个不足，导致 TLL 索引也存在这两个不足：

a)与传统的 Dijkstra 算法一样，Time-dependent Dijkstra 算法需要维护一个最小堆，堆中的元素表示当前已找到的到达顶点的最早时刻，Time-dependent Dijkstra 算法反复从最小堆中取出到达时刻最早的顶点出来扩展路径。如果最小堆用二项堆实现的话，每次从堆中取最小元、调整堆的耗费是 $O(\log n)$，n 表示堆的大小。因为 Time-dependent Dijkstra 需要多次调用堆操作，所以会产生较大的耗费。

b) Time-dependent Dijkstra 算法总是取到达时刻最早的顶点出来扩展路径，没有考虑旅程的完整性，查询所求得的路径有较多的换乘次数。如图 1 所示。假设从站点 A 到 C 有两条路径，线形区分不同的线路：路径 P_1 在时刻 5 从 A 出发，乘坐实线表示的线路直达 C，到达 C 的时刻是 30，途中在时刻 20 的时候路过站点 B。路径 P_2 在时刻 1 从 A 乘坐虚线表示的线路出发，经过若干次换乘后，在时刻 18 到达 B，然后再转乘实线表示的线路，到达 C 的时刻跟 P_1 一样，也是 30。如果要查询在 0 时刻或之后从 A 到 C 的 EAP，Time-dependent Dijkstra 算法会返回 P_2，这是因为，Time-dependent Dijkstra 算法会选取最早到达 B 的路径，即 P_2 从 A 到 B 的那一截路扩展出到达 C 的路径 P_2，而不会选取 P_1 中从 A 到 B 的那一截路扩展生成 P_1。尽管 P_2 和 P_1 的到达时刻一样，但用户会更倾向于选择 P_1，因为 P_1 不需要换乘。如果乘客携带行李，换乘会引起不便，而且会增加延误的风险。

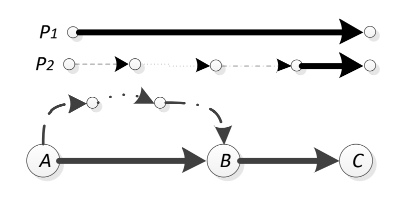


图 1　从 A 到 C 的两条路径

另一方面，在路径规划中，换乘次数是个需要考虑的问题。有部分学者着重研究查找换乘次数少的路径，详见第 6 节的讨论。其中，Delling 等人提出的 RAPTOR[4]是一种快速的查找受换乘次数限制的 EAP 的方法。 RAPTOR 并不采用图结构表示交通数据，而是用一个数组记录旅程数据，查询时直接通过扫描数组求得 EAP。一条旅程记录了一趟交通工具按预定线路行驶，路过每个站点的到达时刻和离开时刻。RAPTOR 从起始站点 s 开始，扫描路过 s 的旅程，找到从 s 可达的站点，再从 s 可达的站点 u 出发，扫描路过 u 的旅程，发现更多可达的站点，如是重复直到从 s 到网络中其他的可达站点的 EAP 都求出来为止。RAPTOR 的不足之处在于：a) RAPTOR 要求得从 s 到 d 的 EAP 的话，必须先求出从 s 到其余所有可达站点的 EAP，显然当中存在不必要的计算;b) RAPTOR 并没有对公交数据做预处理建立索引，所以查询速度远比不上 TTL。但是，由于 RAPTOR 是通过扫描旅程求 EAP 的，保留了旅程的完整性，跟 Time-dependent Dijkstra 相比，能在一定程度上倾向于找到换乘次数较少的路径，这一点将在第 4.2 节详细说明。

本文结合 TTL 和 RAPTOR 的优点提出了一种 TAIL(trip based time labelling) 索引求解 EAP、LDP 和 SDP。与 TTL 相比，TAIL 并不是在图结构上建立索引，而是在表示旅程数据的数组上建立索引，其核心算法采用了基于旅程扫描的算法，弥补了“大量堆操作”与“生成的路径有较多的换乘次数”的两个不足；与 RAPTOR 相比，TAIL 利用索引做查询，比从零信息量上做查询快。本文提出一种基于旅程的查询方法，该方法对 RAPTOR 加以修改，并提出求 LDP 的方法，使之能适用于 TAIL 索引的构建中。实验表明，TAIL 与 TTL 有相仿的索引大小及查询时间，建立索引时间较短，且生成的路径有较少的换乘次数。

1　问题定义

设 S 表示站点(stop)的集合。站点指乘客登上或离开交通工具的地方，例如公共巴士站、地铁站等等。设 R 表示线路(Route)的集合。线路表示一辆交通工具预定的行驶站点的序列，例如公交车线路、地铁线路等等。设 $r \in R$，$r = \langle s_1, s_2, s_3, \cdots s_k \rangle$，其中 $s_1, s_2 \cdots, s_k \in S$，表示线路 r 从站点 s_1 出发，依次途径 $s_2, s_3, \cdots$ 直到终点 s_k。在现实生活中，一条线路一般分两个行驶方向，例如深圳地铁 1 号线分成从机场东往罗湖方向和从罗湖往机场东方向。在本文中，不失一般性，把一条线路的两个行驶方向当作两条线路处理，因为它们途径的站点序列不一样。另一方面，在现实生活中，线路上的站点允许重复出现。在本文中，为了表述清晰，如果线路上出现重复的站点，则按重复站点的出现位置将线路分成多截，当作多条线路处，于是保证了线路上的站点各不相同。

旅程对应一条线路的一次具体行驶，包含每个站点的到达时刻和出发时刻。一趟旅程对应一条线路，一条线路包含多趟旅程。用符号 $\Gamma(r) = \{t_1, t_2, \cdots, t_k\}$ 表示线路 r 包含了旅程 $t_1, t_2, \cdots, t_k$。分别用符号 $\pi_{arr}(t,s)$ 和 $\pi_{dept}(t,s)$ 表示旅程 t 上在站点 s 的到达时刻和出发时刻。不失一般性，本文采用正整数表

示时刻。$\pi_{arr}(t,s)$ 和 $\pi_{dept}(t,s)$ 满足条件 $\pi_{arr}(t,s) \le \pi_{dept}(t,s)$ 。t 上的第一个站点的到达时刻和最后一个站点的出发时刻无定义。

图 2 是一个公交网络的例子，用不同的线形来区分线路。表 1 显示了图 2 对应的各条线路的时间表，表中列出了每个站点的(到达时刻，出发时刻)。

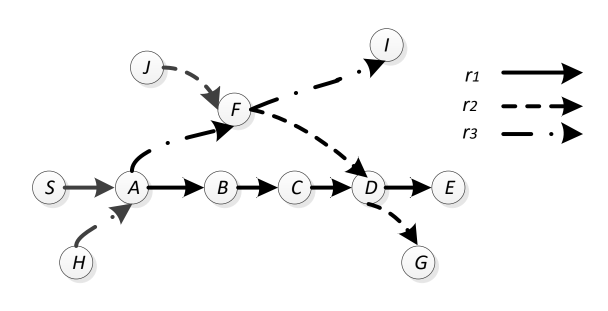


图 2 一个公交网络

定义 1　旅程的≺关系。设 r 是一条线路，$t_1,t_2 \in \Gamma(r)$ 是行驶线路 r 的两条不同的旅程。定义 $t_1 \prec t_2$，如果满足以下两个条件：a)对于 r 上的除了第一个站点以外的所有站点 s，有 $\pi_{arr}(t_1,s) \le \pi_{arr}(t_2,s)$，且 b)对于 r 上的除了最后一个站点以外的所有站点 s，有 $\pi_{dept}(t_1,s) \le \pi_{dept}(t_2,s)$ 。

$t_1 \prec t_2$ 的意思即是说，若在前一个站点 t_1 比 t_2 先到达(或离开)，在下一个站点也必定有 t_1 比 t_2 先到达(或离开)。≺关系也称为不超越关系(no over-taking)。在现实的公交网络中，一般情况下同一条线路下的旅程之间满足≺关系。但是，堵车等因素会破坏≺关系，即在前一个站晚到达（或出发）的旅程在后续的站点中会早到达（或出发）。在本文中，若在同一条线路下存在旅程 t_1、t_2，不满足≺关系，则将 t_1 和 t_2 当作是属于两条不同的线路处理。于是≺关系是 $\Gamma(r)$ 上的一个全序。

表 1 图 2 中公交网络时间表

线路	旅程	途径站点与各站点的(到达，出发)时刻					
r_1		S	A	B	C	D	E
	t_1	(#,1)	(4,4)	(8,9)	(12,12)	(15,15)	(19,#)
	t_2	(#,4)	(7,8)	(14,15)	(20,20)	(23,23)	(27,#)
	t_3	(8,8)	(11,11)	(16,17)	(21,21)	(24,24)	(28,#)
r_2		J	F	D	G		
	t_4	(#,8)	(11,11)	(14,15)	(17,#)		
r_3		H	A	F	I		
	t_5	(#,4)	(7,8)	(10,10)	(12,#)		

定义 2　时间表。$\mathcal{T}=(S,R,T)$ 是一个时间表，其中 S 是站点的集合，$R=\{r_1,r_2,\cdots,r_m\}$ 是线路的集合，$T=\{\{t_{11},t_{12},\cdots\},\{t_{21},t_{22},\cdots\},\cdots,\{t_{m1},t_{m2},\cdots\}\}$ 是旅程的划分。T 下的每个子集与 R 中的元素一一对应，即 $\forall r \in R$，$\Gamma(r) \in T$ 。不失一般性设 T 下的每个子集内的旅程已按≺关系排序，即对于 $1 \le i \le m$，$t_{i1} \prec t_{i2} \prec \cdots$。

定义 3　路径。设 s 是起始站点，d 是目标站点。从 s 到 d 的路径定义为 $P=s_1@t_1,s_2@t_2,\cdots,s_k@t_k,s_{k+1}$，其中 $s_1=s$，$s_{k+1}=d$，对于 $2 \le i \le k$，s_i 是旅程 t_{i-1} 和 t_i 上的站点。P 表示在站点 s_1 上车乘坐旅程 t_1，到站点 s_2 转乘旅程 t_2，……到站点 s_k 转乘旅程 t_k，到达站点 s_{k+1}。在换乘站点上满足时间约束 $\pi_{arr}(t_{i-1},s_i) \le \pi_{dept}(t_i,s_i)$，即换乘时前一趟旅程的到达站点的时刻不能晚于后一趟旅程的出发时刻。称 P 是长度为 k 趟旅程的路径。P 的出发时刻记为 $\pi_{dept}(P)$，取值 $\pi_{dept}(t_1,s_1)$ 。P 的到达时刻记为 $\pi_{arr}(P)$，取值 $\pi_{arr}(t_k,s_{k+1})$ 。P 的耗时记为取值 $\pi_{arr}(P)-\pi_{dept}(P)$ 。本文研究的最早到达路径、最晚到达路径和最短耗时路径定义如下。

定义 4　最早到达路径(EAP)。给定时间表 $\mathcal{T}$、起始站点 s、目标站点 d、时刻 π，EAP 指在所有从 s 到 d 的、出发时刻不早于 π 的路径中，具有最早到达时刻的那条路径。

定义 5　最晚出发路径(LDP)。给定时间表 $\mathcal{T}$、起始站点 s、目标站点 d、时刻 π'，LDP 指在所有从 s 到 d 的、到达时刻不晚于 π' 的路径中，具有最晚出发时刻的那条路径。

定义 6　最短耗时路径(SDP)。给定时间表 $\mathcal{T}$、起始站点 s、目标站点 d、时刻 π 和 π'，SDP 指在所有从 s 到 d 的、出发时刻不早于 π、到达时刻不晚于 π' 的路径中，具有最短耗时的那条路径。

表 2 给出本文中常用的符号及解释。

表 2 文中常用的符号和解释。

符号	含义
$\Gamma(r)=\{t_1,t_2,\cdots,t_k\}$	行驶线路 r 的旅程的集合
$\pi_{arr}(t,s)$、$\pi_{dept}(t,s)$	旅程 t 上在站点 s 的到达时刻和出发时刻。
$t_1 \prec t_2$	t_1 不超越 t_2，见定义 1。
$\mathcal{T}=(S,R,T)$	时间表，见定义 2。
$P=s_1@t_1,s_2@t_2,\cdots,s_k@t_k,s_{k+1}$	路径 P，见定义 3。
$\pi_{dept}(P)$、$\pi_{arr}(P)$	P 的出发时刻、到达时刻
$L_+(u)$ 和 $L_-(u)$	TAIL 索引中的站点 u 的标签集合，见定义 9。

2　TAIL 索引

2-hub-labeling 是路网（图）上求解最短路径的一种高效索引，TTL 引入 2-Hub-Labeling 的思想，对含有时间信息的图建立索引。TAIL 也引入 2-Hub-Labeling 的思想，但并不是对图建立索引，而是对时间表 $\mathcal{T}=(S,R,T)$ 建立索引。与 TTL 相似，TAIL 对每个站点 u 计算两个标签集合：$L_+(u)$ 和 $L_-(u)$ 。$L_+(u)$ 中的每个标签对应一条从 u 出发到达某个站点的路径；对称地，$L_-(u)$ 中的每个标签对应一条从某个站点出发到达 u 的路径。对于任意的从起点 s 到目标 d 的 EAP、LDP 和 SDP 查询，TAIL 可以从 $L_+(s)$ 和 $L_-(d)$ 中找到标签拼接成一条路径得到相应的

解。这种方法利用预先计算好的路径信息查找路径，避免在原数据集上从零信息开始搜索，大大提高了查询的效率。

TAIL 的构建基于一个站点的全序关系，全序关系用来衡量站点在公交网络中的重要程度。在公共交通网络中，交通枢纽很重要，因为相距较远的两个地方通常在交通枢纽换乘。本文用 $o(u)$ 表示将站点按重要程度从高到低排序后，u 在序列中的位置。若 $o(u)<o(v)$，表明站点 u 的重要程度排在站点 v 的前面，即 u 的等级比 v 高。o 的选取并不影响查询结果的正确性，但会影响索引的大小，进而影响查询效率。下文将在第 4.4 节中讨论 o 的计算。

TAIL 的索引并没有保存所有从 u 出发和到达 u 的路径，只保留部分基本路径。

定义 7　路径支配。设 P_1 和 P_2 分别是两条从站点 u 到站点 d 的路径。称 P_1 支配 P_2，如果 $\pi_{dept}(P_1)\geq\pi_{dept}(P_2)$，且 $\pi_{arr}(P_1)\leq\pi_{arr}(P_2)$。

定义 7 的意思是：如果 P_1 的出发时刻不早于 P_2 的，且到达时刻不晚于 P_2 的，则 P_1 比 P_2 更优。若 P_2 在一条 EAP/LDP/SDP 上，则用 P_1 代替 P_2 那截子路，也能得到 EAP/LDP/SDP。所以索引只需考虑不被支配的路径即可。

定义 8　基本路径。$P=s_1@t_1,s_2@t_2,\cdots,s_k@t_k,s_{k+1}$ 是一条基本路径，如果满足以下两个条件：a)站点等级约束,P 途径的站点中，s_1 或 s_{k+1} 的等级最高；b)支配约束,不存在另一条从 s_1 到 s_{k+1} 的路径 P'，P'支配 P。

假设图 2 中的站点的重要程度依次为:DAFSBCEJGHI。$B@t_1,D$ 是一条基本路径；$A@t_2,D$ 不是基本路径，因为存在路径 $A@t_5,F@t_4,D$ 支配 $A@t_2,D$。

定义 9　TAIL 索引。给定时间表 $\mathcal{T}=(S,R,T)$，TAIL 索引预先对每个站点 $u\in S$ 计算两个标签集合 $L_+(u)$ 和 $L_-(u)$，满足以下四个条件：

a) $L_+(u)$ 中的每个标签 $\langle v,\pi_{dept},\pi_{arr},s_{nei}\rangle$ 对应一条从 u 到 v 的基本路径 P，且有 $o(v)<o(u)$。其中 P 在时刻 π_{dept} 出发，在时刻 π_{arr} 到达。s_{nei} 是 P 上继 u 之后的第一个换乘站点。

b) $L_-(u)$ 中的每个标签 $\langle v,\pi_{dept},\pi_{arr},s_{nei}\rangle$ 对应一条从 v 到 u 的基本路径 P，且有 $o(v)<o(u)$。其中 P 在时刻 π_{dept} 出发，在时刻 π_{arr} 到达。s_{nei} 是 P 上到达 u 的前一个换乘站点。

c)对于任意的从站点 s 到站点 d 的 EAP、LDP 或 SDP 查询 q，如果存在解的话，以下三种情况之一必成立：(a)在 $L_+(s)$ 中存在标签 $\langle d,\pi_{dept},\pi_{arr},\cdot\rangle$ [1]对应 q 的解；(b)在 $L_-(d)$ 中存在标签 $\langle s,\pi_{dept},\pi_{arr},\cdot\rangle$ 对应 q 的解；或(c)在 $L_+(s)$ 中存在标签 $\langle w,\pi_{dept},\pi_{arr},\cdot\rangle$ 对应路径 P_+，在 $L_-(d)$ 中存在标签 $\langle w,\pi'_{dept},\pi'_{arr},\cdot\rangle$ 对应路径 P_-，满足 $\pi_{arr}\leq\pi'_{dept}$，使得 P_+ 和 P_- 拼接成的路径构成 q 的解。

d) $L_+(u)$ 中的标签按目标顶点的等级序 o 为第一关键字、出发时刻为第二关键字升序排序。即是说，设 $l=\langle v,\pi_{dept},\pi_{arr},\cdot\rangle$ 和 $l'=\langle v',\pi'_{dept},\pi'_{arr},\cdot\rangle$ 都是 $L_+(u)$ 中的标签，l 排在 l'之前，如果 $o(v)<o(v')$，或 $o(v)=o(v')$ 且 $\pi_{dept}<\pi'_{dept}$。对 $L_-(u)$ 中的规则标签也按同样的规则排序。

对于定义 9 的第 d)点，在 $L_+(u)$ 中的标签 l 和 l'，若 $o(v)=o(v')$ 且 $\pi_{dept}<\pi'_{dept}$，由基本路径的支配性可得，必有 $\pi_{arr}<\pi'_{arr}$。即是说，$L_+(u)$ 中具有相同目标点的标签，按出发时刻和到达时刻的升序排列。对 $L_-(u)$ 同理。

表 3 列出了对应表 1 的例子中的标签集合 $L_+(S)$ 和 $L_-(E)$ 的内容。

表 3 标签集 $L_+(S)$ 和 $L_-(E)$

集合	标签
$L_+(S)$	$l_1\langle D,4,14,A\rangle$ $l_2\langle D,8,24,A\rangle$ $l_3\langle A,1,4,S\rangle$ $l_4\langle A,4,7,S\rangle$ $l_5\langle A,8,11,S\rangle$
$L_-(E)$	$l_6\langle D,15,19,E\rangle$ $l_7\langle D,23,27,E\rangle$ $l_8\langle D,24,28,E\rangle$

3　查询算法

定义 9 的第 c)点给出了查询的思路。对于从站点 s 到站点 d 的、出发时刻不早于 π、到达时刻不晚于 π' 的 SDP 查询 q，算法在 TAIL 索引的 $L_+(s)$ 中找到出发时刻 $\geq\pi$ 的标签 $l_+=\langle u,\pi_{dept},\pi_{arr},\cdot\rangle$，和在 $L_-(d)$ 中找到到达时刻 $\leq\pi'$ 的标签 $l_-=\langle u',\pi'_{dept},\pi'_{arr},\cdot\rangle$，如果 $u=u'$ 且 $\pi_{arr}\leq\pi'_{dept}$，则 l_+ 和 l_- 拼接的路径是 q 的候选解。算法找出所有候选解，再从中选出符合查询 q 的解。这个求解过程类似数据库的连接操作。对于 EAP 查询，取 $\pi'=\infty$；对于 LDP 查询，取 $\pi=0$。由于标签集合内的标签已排好序，所以可以通过线性扫描 $L_+(s)$ 和 $L_-(d)$ 的集合找出所有候选解。例如查询从 S 到 E 的、出发时刻不早于 3、到达时刻不晚于 30 的 SDP，对应表 3，首先在 $L_+(S)$ 扫描第一个标签 l_1，在 $L_-(E)$ 中扫描第一个标签 l_6，l_1 和 l_6 能构成路径，得到候选解，用 l_1+l_6 表示。由于 $L_-(E)$ 中起点为 D 的标签(l_6,l_7,l_8)是按到达时刻升序排列的，因此没必要再配对 l_1 和 l_7,l_8，因为它们不可能生成更优的解。接下来，扫描 $L_+(S)$ 中的下一个标签 l_2，同时也扫描 $L_-(E)$ 中的下一个标签 l_7。因为 l_2 和 l_7 不能构成路径，所以只能再扫描 $L_-(E)$ 中的下一个标签 l_8，得到又一条候选路径，用 l_2+l_8 表示。至此 $L_-(E)$ 的标签扫描完毕，查询结束。在 l_1+l_6 和 l_2+l_8 两者中，l_1+l_6 的耗时最短，为查询 q 的解。算法详情请参考文[1]。

上述算法只能得到解的标签，设来自 $L_+(s)$ 的标签记为 l_+^{opt}，来自 $L_-(d)$ 的标签记为 l_-^{opt}，需要用标签重构回包含换乘详细信息的完整路径。因为 TAIL 与 TTL 生成基本路径的方法不同，

[1]对标签中暂不关注的分量的值用点表示，下同。

所以对路径标记的方法也不同。设 l_+^{opt} 表示的路径的目标点(即 l_-^{opt} 表示的路径的起始点)是 v，则查询 q 生成的路径路过的等级最高的站点是 v。以 l_+^{opt} 为例解释重构路径的过程。假设 l_+^{opt} 对应路径 $P=s@t_1,w@t_2,\cdots,v$，l_+^{opt} 记录下站点第一个换乘 w。例如表 3 中的标签 $l_1\langle 4,14,A\rangle$ 表示的路径是 $S@t_2,A@t_5,F@t_4,D$，所以 l_1 记录下站点 A。可证明 $L_+(w)$ 包含一条标签 l_{sub}，l_{sub} 对应一条从 w 到 v 的、出发时刻不早于 $\pi_{arr}(t_1,w)$、到达时刻不晚于 $\pi_{arr}(P)$ 的基本路径。这是因为：

a) P 是 $L_+(s)$ 中的标签对应的基本路径，所以 P 路过的站点中，v 的等级最高，路径 P 的后缀 $P_{sub}=w@t_2,\cdots,v$ 满足站点的等级约束。

b) P_{sub} 是一条出发时刻不早于 $\pi_{arr}(t_1,w)$、到达时刻等于 $\pi_{arr}(P)$ 的路径。若不存在路径 P' 支配 P_{sub}，结合(1)，则 P_{sub} 在 $L_+(w)$ 内。否则，设存在路径 P' 支配 P_{sub}，证明 l_{sub} 对应 P'。分两种情况讨论：(a)如果 P' 的站点满足等级约束，则 P' 是基本路径，所以 l_{sub} 对应 P';(b) 如果 P' 的站点不满足等级约束，即 P' 上路过某个站点的等级比 v 高；另一方面，在 P 中用 P' 代替 P_{sub}，与 l_-^{opt} 能拼接上路径，构成的路径也是查询 q 的解，这与"查询 q 生成的路径路过的等级最高的站点是 v"矛盾。

因此，已知 l_+^{opt} 的第一个换乘点是 w，必定能在 $L_+(w)$ 下找到标签重构 $w@t_2,\cdots,v$ 这段子路，这只需要在 $L_+(w)$ 下调用一次二分查找即可。对 l_-^{opt} 的重构同理。假设查询结果是长度为 k 趟旅程的路径，则 l_+^{opt} 和 l_-^{opt} 最多需要 k 次二分查找即可重构完整路径。

4　构建索引

4.1　基于旅程扫描的查询算法

RAPTOR[4]的全称是 Round-bAsed Public Transit Optimized Router，顾名思义，算法的思想是"一轮接一轮"地查找最早到达路径，所谓"轮"指一次从上车到下车的乘坐过程。第一轮，算法扫描途径起始站点的每一条线路，发现从起始站点直达的站点，更新它们的最早到达时刻，把它们的集合记为 Q_1。换句话说，当前找到的到达 Q_1 中的站点的路径，都是长度为 1 趟旅程的路径。接下来，RAPTOR 如是重复以下步骤：令 Q_k 表示经过 k 轮乘坐后到达的、且到达时刻比经过 $i(i<k)$ 轮乘坐的更早的站点的集合，即到达 Q_k 中的站点 u 的路径，是长度为 k 趟旅程的路径，而且比长度为 $i(i<k)$ 趟旅程的路径都早到达。扫描途径 u 中的站点的线路，有可能发现 $k+1$ 轮到达的、到达时刻更早的站点，生成集合 Q_{k+1}。换句话说，RAPTOR 从 $k=1$ 开始，先求出长度为 k 趟旅程的最早到达路径，再求出长度为 $k+1$ 趟旅程的最早到达路径。这个过程有点类似图的广度优先遍历。

假设到达站点 v 的最早到达路径需要经过站点 u，那么，在 u 的 EAP 计算出来之前，用站点 u 去扩展路径的工作是徒劳的。譬如在图 2 中，从站点 S 到达 E 的出发时刻不早于 3 的路径有 $S@t_2,A@t_5,F@t_4,D@t_1,E$ 和 $S@t_2,E$，其中前者是 EAP。到达 E 的路径需要途径 D，而 D 的最早到达时刻会影响 E 的最早到达时刻，进而会影响从 E 出发的路径（如果公交网络中还有从 E 出发的路径的话）。RAPTOR 需要等到第 3 轮才能找到到达 D 的 EAP $S@t_2,A@t_5,F@t_4,D$，即是说，在第 2、3 轮扩展站点 D 及 D 出发的路径都是徒劳。因此，本文的算法去掉扩展路径时的轮数限制，仅用集合 Q 记录到达时刻得到更新的站点的集合，初始时 Q 仅包含起始站点。取 Q 中的站点 u 出来扩展路径，若发现新扩展的路径能更早地到达站点 v，则将 v 加进 Q，如此重复直到 Q 为空。这样能部分减少无谓的路径扩展。

算法 1 描述了查找从 s 到 d 的、出发时刻不早于 π 的 EAP 的过程。算法用 $\tau(u)$ 记录当前找到的到达站点 u 的最早时刻。初始时，s 的最早到达时刻设为 π，其它站点的最早到达时刻设为 ∞(第 1 行)。算法用符号 $\lambda_{r,u}$ 记录线路 r 上的站点 u 被扫描过的最早的旅程是哪一条。在后续计算过程中，当扫描 r 上从 u 出发的旅程时，只需要扫描 $\prec\lambda_{r,u}$ 的旅程即可：因为 $\lambda_{r,u}$ 的取值意味着算法已经用旅程 $\lambda_{r,u}$ 检查 r 上在 u 之后的站点 v，使得的 v 最早到达时刻不会晚于 $\pi_{arr}(\lambda_{r,u},v)$。因此，当再次从 r 上的站点 u 开始扫描旅程时，只有的 $\prec\lambda_{r,u}$ 旅程才有可能使得 $\tau(v)$ 变小。算法通过记录 $\lambda_{r,u}$ 限制旅程被重复扫描。初始时 $\lambda_{r,u}$ 设为 $t_\top$，它表示旅程的最大元，即对所有旅程 t，均有 $t\prec t_\top$(第 2 行)。

算法 1. *FindEAP*$(\mathcal{T},s,\pi)$

输入：起始站点 s，时刻 π。

输出：在不早于 π 时刻从 s 出发到达其余站点的 EAP 的到达时刻。

1. 令 $\tau(s)=\pi$,对于 $s'\in S-\{s\}$，令 $\tau(s')=\infty$。
2. 对于所有的 r，设 u 是 r 上的站点。令 $\lambda_{r,u}=t_\top$。
3. 令 $\sigma(s)=nil$。
4. 令集合 $Q=\{s\}$。
5. while　Q 非空
6. 　　从 Q 取出一个站点 u。
7. 　　foreach　途径 u 的线路 r
8. 　　　　设 p 是满足 $p\prec\lambda_{r,u}$ 且 $\pi_{dept}(p,u)\ge\tau(u)$ 的最早的旅程。如果 p 不存在，转 7。
9. 　　　　扫描 p，从 u 点开始依次访问后继站点 v
10. 　　　　　　if　$\lambda_{r,v}$ 等于 p，停止扫描 p，转 7。
11. 　　　　　　if　$\pi_{arr}(p,v)<\tau(v)$
12. 　　　　　　　　令 $\tau(v)=\pi_{arr}(p,v)$，$\sigma(v)=u$
13. 　　　　　　　　令 $\lambda_{r,v}=p$。
14. 　　　　　　　　将 v 加进 Q。
15. 返回所有站点的最早到达时刻 $\tau(\cdot)$。

算法用 $\sigma(u)$ 记录在到达 u 之前的换乘站点，初始时 $\sigma(s)$ 设为空(第 3 行)。接下来，算法反复从集合 Q 取出站点扩展路径。在扩展站点转乘的路径时，对于每一条路过 u 的线路 r(第 7 行)，如果存在比之前扫描过的更早的旅程 p，满足 p 上 u 的出发时刻不早于 u 的到达时刻(第 8 行)，则沿线更新 u 之后的站点 v 的到达时刻(第 9~14 行)。如果发现乘坐旅程 p 到达 v 的时刻比之前找到的更优(第 11 行)，则更新 $\tau(v)=\pi_{arr}(p,v)$，并

记录 v 是通过在 u 点换乘到达的(第 12 行)，记录下线路 r 上站点 v 扫描过的最早旅程是 p(第 13 行)，将 v 加进 Q (第 14 行)。当再没有站点的最早到达时刻有更新时，算法结束，返回站点的最早到达时刻 $\tau(\cdot)$ （第 15 行)。

假设在图 2 的例子中，查找从站点 S 到站点 E 的、出发时刻不早于 3 的 EAP，表 4 列出了求解过程。第一列 $u@t$ 表示扫描旅程 t，从站点 u 开始沿线更新 u 之后的站点的最早到达时刻。第二列表示扫描 t 后生成的集合 Q，第三列列出扫描 t 后更新了的站点的最早到达时刻。表 4 仅列出算法扫描过的的旅程。例如在第 2 行处理完 A 站点之后，该从 Q 取出 B。因为路过 B 的线路 r_1，满足出发时刻≥14(B 的到达时刻)的旅程还是 t_2，而之前在扫描 $S@t_2$ 的时候已经记录下 $\lambda_{r_1,B}=t_2$，所以 t_2 不会重复被扫描。

表 4　在图 2 中查找从 S 到 E、出发时刻不早于 3 的 EAP 的过程

$u@t$	Q	更新的各站点的最早到达时刻
$S@t_2$	{A,B,C,D,E}	A:7,B:14,C:20,D:23,E:27
$A@t_5$	{B,C,D,E,F,I}	F:10,I:12
$F@t_4$	{I,D,G}	D:14, G:17
$D@t_1$	{G,E}	E:19

对称地，可得到算法 $FindLDP(\mathcal{T},d,\pi)$ 查询从其余站点出发、在 π 时刻到达站点 d 的最晚出发路径。$FindLDP$ 从目标站点 d 开始反向扩展路径，求得每个站点 u 要在 π' 时刻到达 d 的话，最晚出发时刻 $\tau(u)$。具体地，初始时集合 $Q=\{d\}$，每次从 Q 取出一个站点 u，扩展每条途径 u 的线路 r；找到 r 下的满足 $\pi_{arr}(p,u)\le\tau(u)$ 最晚的旅程 p，从 u 开始，沿着 p 反向扫描 u 的前驱站点 v，更新 $\tau(v)$。

4.2　换乘次数

由于算法 1 通过扫描旅程的方式生成路径，所以在一定程度上保持了旅程的完整性。在算法 1 中，换乘次数少的路径先被发现，换乘次数多的路径后被发现，如果换乘次数多的路径不能改进目标点的最早到达时刻，则仍保留换乘次数少的路径，进而减少了路径的换乘次数。以图 2 和表 1 的公交网络为例解释，假设 t_4 到达和离开 D 的时刻改成 16 和 16，求解在 8 时刻或之后从 A 出发的 EAP。TTL 找到的路径是 $A@t_5,F@t_4,D@t_2,E$，到达时刻是 27。TAIL 第一步扫描路过 A 的旅程有 t_2 和 t_5，在扫描 t_5 时已经得到路径 $A@t_5,E$ （换乘次数少的路径)，以及 $\tau(E)=27$。在随后的扫描，得到路径 $A@t_5,F@t_4,D$，$\tau(D)=16$。再次扫描路过 D 的旅程时，t_2 已在 D 点被扫描过；另一方面，通过更多次换乘、到达时刻也比 $\pi_{arr}(t_2,D)$ 早的路径 $A@t_5,F@t_4,D$ 沿 r_1 线能走的最早的旅程仍然是 t_2，即不能改进 E 的到达时刻，所以到达 E 的路径 $A@t_5,E$ 仍然保留，不会被替换。

4.3　生成标签集合

TAIL 索引需要计算时间表 $\mathcal{T}$ 中的所有基本路径，本节讨论如何不遗漏地、高效地找出所有基本路径。设 S 中站点的等级从高到低排序依次为 $s_1,s_2,\cdots,s_n$，即对于站点 s_i，$s_1,\cdots,s_{i-1}$ 的等级比 s_i 的高，$s_{i+1},\cdots,s_n$ 的等级比 s_i 的低。设 Π 表示时间表 $\mathcal{T}$ 中 s_i 的出发时刻的集合。对于所有的 $\pi\in\Pi$，在 π 时刻从 s_i 出发的 EAP 可分成两类：第一类路径不途径集合 $\{s_1,\cdots,s_{i-1}\}$ 中的站点，它们可能是基本路径；第二类路径途径 $\{s_1,\cdots,s_{i-1}\}$ 中的站点，它们违反了站点的等级约束，不是基本路径。对于第二类路径 P，设 P_{pre} 是 P 上从起点 s_i 到 s($s\in\{s_1,\cdots,s_{i-1}\}$)的前一个站点的那截路，则 P_{pre} 可能是基本路径。换句话说，在调用 $FindEAP(\mathcal{T},s_i,\pi)$ 求从 s_i 出发的 EAP 的时候，如果在扫描旅程时($FindEAP$ 的第 9 行)遇到 $s\in\{s_1,\cdots,s_{i-1}\}$ 的站点 s，则不需要再扫描 s 之后的站点 v，因为到达 v 的路径需要路过 s，它不可能是基本路径。算法 2 描述了建立索引的过程，第 4~11 行描述了求从 s_i 出发的基本路径的具体过程。

算法 2.　$BuildIndex(\mathcal{T},o)$
输入: 时间表 $\mathcal{T}$=(S,R,T) 和站点序 o
输出: TAIL 索引。

1. 对于所有站点 u, $L_+(u)$ 和 $L_-(u)$ 都初始化为 $\varnothing$。
2. for　$i=1,2,\cdots,n$
3. 　设 s_i 是序为 i 的站点，即 $o(s_i)=i$。
4. 　设 Π 表示所有从 s_i 出发的时刻的集合，即 $\Pi=\{\pi_{dept}(t,s_i)\mid s_i$ 是旅程 t 途径的站点}。
5. 　令 $\tau(s_i)=\pi$,对于 $s'\in S-\{s_i\}$，令 $\tau(s')=\infty$
6. 　对于所有的 r，设 u 是 r 上的站点。令 $\lambda_{r,u}=t_\top$。
7. 　按时刻从晚到早的顺序访问每个 $\pi\in\Pi$
8. 　　调用 $FindEAP(\mathcal{T},s_i,\pi)$，作以下两个改动：(1) 从第 3 行开始调用；(2) 第 10 行增加停止条件：如果 $\lambda_{r,v}$ 等于 p，或 $o(v)<o(s_i)$，停止扫描 p
9. 　　foreach 站点 s: $o(s)>o(s_i)$
10. 　　　令 $l=\langle s,\pi,\tau(s),\sigma(s)\rangle$
11. 　　　if l 均满足以下两个条件：(1) 在 $L_-(s)$ 中不存在标签 $l'=\langle s,\cdot,\pi_{arr},\cdot\rangle$，使得 $\pi_{arr}\le\tau(s)$；(2) 不存在 $l_+=\langle v,\pi^+_{dept},\pi^+_{arr},\cdot\rangle\in L_+(s_i)$，，$l_-=\langle v,\pi^-_{dept},\pi^-_{arr},\cdot\rangle\in L_-(s)$，$\pi^+_{arr}\le\pi^-_{dept}$，且 $[\pi^+_{dept},\pi^-_{arr}]\subset[\pi,\tau(s)]$ 在，则将 l 添加进 $L_-(s)$ 中。
12. 　重复第 4~11 行，设 Π 是所有到达 s_i 的时刻的集合，即 $\Pi=\{\pi_{arr}(t,s_i)\mid s_i$ 是旅程 t 途径的站点}。对 Π 中的所有时刻 π，按从晚到早的顺序调用 $FindLDP(\mathcal{T},s_i,\pi)$，与第 8 行做类似的改动，生成标签集合。
13. 返回所有站点的标检集合。

在求从 s_i 出发的基本路径的时候，一种直观的做法是求出在 Π 中任意时刻 π 从 s_i 出发的EAP，再把被支配的EAP去掉，但这样作会产生很大的计算量。留意到在 π 时刻出发的路径仅可能被 π' ($\pi'\ge\pi$)时刻出发的路径支配，因此，可以按照 Π 中的时刻从晚到早的顺序生成 EAP，新生成的 EAP 仅可能被已生成的 EAP 支配，于是通过检查已生成的 EAP 判断新生成的 EAP 是否是基本路径，决定是否将它加进标签集合。

设 π_{max} 是 Π 中的最大元。在调用完 $FindEAP(\mathcal{T},s_i,\pi_{max})$ 之后，$\tau(s)$ 即为在 π_{max} 时刻从 s_i 到 s 的最早到达时刻，$\lambda_{r,u}$ 表示途径线路 r 上的站点 u 的最早的旅程是哪条。在下一趟求在 π' ($\pi'<\pi_{max}$)时刻出发的 EAP 的时候，可以延用 $\tau(s)$ 和 $\lambda_{r,u}$ 的值。

这是因为：已找到路径 P 在 π_{max} 时刻从 s_i 出发、在 $\tau(s)$ 时刻到达 s，那么，若在 π' 时刻从 s_i 出发、到达 s 的路径 P'的到达时刻 $\geq\tau(s)$，则 P'被 P 支配，于是没必要再扩展 P'，因为扩展 P'不会得到比扩展 P 更优的路径。对 $\lambda_{r,u}$ 的初始化同理。因此，算法 2 仅在第 5、6 行对 $\tau(s)$ 和 $\lambda_{r,u}$ 做初始化，随后，按照时刻从晚到早的顺序调用 $FindEAP(\mathcal{T},s_i,\pi)$ （第 7、8 行）。调用完 $FindEAP(\mathcal{T},s_i,\pi)$ 后，检查每个等级比 s_i 低的站点 s(第 9 行)，生成在 π 时刻从 s_i 出发、在 $\tau(s)$ 时刻到达 s 的标签 l(第 10 行)。如果 l 不被 $L_{_}(s)$ 中的标签对应的路径支配(第 11 行的条件 1)，也不被从 s_i 到 s 路过更高等级的点的路径支配(第 11 行的条件 2)，则将 l 加进 $L_{_}(s)$ 中。

算法第 12 行采用与第 4~11 行逆向的方式，调用 $FindLDP$，找出到达 s_i 的 LDP，从中求出基本路径，由于篇幅关系不再赘述。

4.4　站点等级序 o 的计算

尽管站点的等级序 o 的选取不影响查询结果的正确性，但会影响索引所包含的标签数目，进而影响查询效率。要精确计算出 o，使得索引所包含的的标签数最少是一个 NP-难的问题。本文沿用文[1]的方法，对 o 进行启发式计算。如果一个站点 u 在一条 EAP P 上，称 u 覆盖 P。从 u 出发的标签和到达 u 的标签可以生成部分路过点 u 的 EAP。所以，直观地，u 覆盖的 EAP 越多，u 的等级就应该越高，这样 u 出发的和到达 u 的标签就可以生成越多的 EAP。由于 EAP 太多逐一枚举耗费太大，所以采用这样的方法计算站点等级序 o:随机生成一个站点集合 $S_{sub}\subset S$，对 S_{sub} 中的每个站点 u，随机生成一个时刻 π，调用 $FindEAP(\mathcal{T},u,\pi)$ 求出 u 到其余站点的 EAP，这些 EAP 构成一棵以 u 为根的树，这样一共有 $|S_{sub}|$ 棵树。对于每棵树，留意到树上的顶点 v 到 v 的子孙的路径也是 EAP，因此 v 覆盖的 EAP 数目等于以 v 为根的子树所包含的顶点数，包括 v 自身。如是统计各个站点 u 在 $|S_{sub}|$ 棵树覆盖的 EAP 总数。算法首先选出覆盖 EAP 数最多的点 s_1 作为等级最高的站点；然后在每棵树中删除以 s_1 为根的子树，再更新各个站点覆盖的 EAP 数，选择剩下的子树中覆盖 EAP 数目最大的站点 s_2 作为等级次高的站点，依次类推。若 $|S_{sub}|$ 棵树被删除完后，仍有站点的等级未定，站点所在的旅程总数多的等级排在前列。

5　实验

实验采用 C++语言编写测试代码，测试平台是 Centos 7.0，机器配置 CPU Intel(R) Xeon(R) CPU E5-2640 v4，内存 64 GB。实验数据来自 GTFS[5]。GTFS 提供了某些地区的真实的公交数据，数据格式与本文讨论的时间表 $\mathcal{T}$ 的格式一致。GTFS 中对时间采用"时:分:秒"的格式表示，在实验中转换成以秒为单位的整数表示，例如:"9:10:06"用整数 33006 表示。因为部分 GTFS 数据中，同一条线路下的旅程不满足≺关系，所以把这些旅程拆分成不同线路处理。TTL 算法基于图结构，用 GTFS 数据生成图。数据集的大小如表 5 所示，对于每个数据集，TAIL 的站点数等于 TTL 的顶点数。

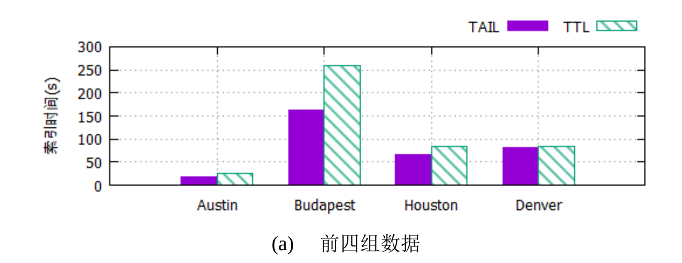
表 5 数据集大小

数据集	TAIL		TTL	
	线路数	旅程数	顶点数	边数
Austin	0.36 K	11.7 K	2.6 K	363K
Budapest	1.8 K	114.7 K	7.2 K	1971 K
Houston	0.5 K	17.4 K	8.9 K	1077 K
Denver	1.3 K	30.7 K	9.3 K	1160 K
Saint Petersburg	2.2 K	261.6 K	7.3 K	2863 K
Rome	1.5 K	100.8 K	9.1 K	3004 K
Toronto	1.7 K	91.3 K	10.3 K	3755 K
Berlin	15.7 K	169.3 K	39.9 K	3377 K

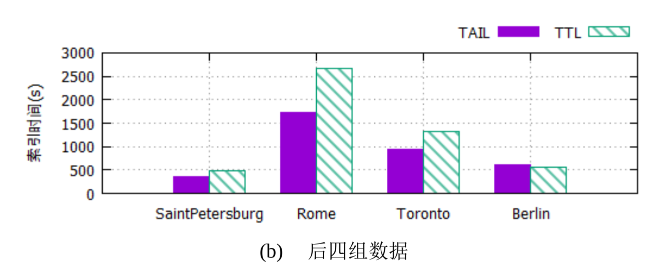
实验从建立索引的时间、生成索引的大小、查询时间和生成路径的换乘次数比较 TAIL 和 TTL 索引。因为 TAIL 和 TTL 索引的思想相似，区别在于生成索引的算法不一样，所以 TAIL 和 TTL 索引大小、查询时间相仿，由于篇幅关系本文不再罗列索引大小和查询时间的数据。下面从建立索引的时间和生成路径的长度对比 TAIL 和 TTL。

5.1　建立索引的时间

TAIL 和 TTL 索引的建立都需要基于顶点/站点的等级序 o。实验首先对每个数据集生成等级序 o，然后对 TAIL 和 TTL 生成索引。索引生成两遍，时间取两次实验的平均值。TAIL 和 TTL 生成索引的时间如图 3 所示，纵轴显示以 s 为单位。



(a)　前四组数据



(b)　后四组数据

图 3 建立索引的时间

从图 3 可得，TAIL 建立索引的时间总体比 TTL 的小，原

因在于两点：a) TAIL 反复用 O(1)的时间从集合取一个站点出来扩展路径，而 TTL 需要反复从最小堆用 O(log n)的时间取一个站点；b) 在按出发时刻从最晚到最早的顺序求从站点 s_i 出发的基本路径的时候，TAIL 利用了在 π 时刻计算的 $\lambda_{r,s}$ 的值避免重新搜索旅程。

此外，TAIL 建立索引所花的时间与平均一条线路包含旅程数和平均一条线路上的站点数(即站点数/线路数)相关。以求从 s_i 出发的 EAP 为例解释其原因。首先，$FindEAP(\mathcal{T},s,\pi)$ 第 8 行在同一条线路下查找满足时间约束的最早的旅程来更新站点的最早到达时刻。如果一条线路包含的旅程数越多，此步骤所排除的旅程数(排除那些不能生成 EAP 的旅程)也越多，因此建立索引的时间越短，例如数据集 Budapest 和 Rome。其次，平均一条线路的站点数越多，TAIL 和 TTL 建立索引的时间差就越明显；而平均一条线路上的站点数越少，TAIL 和 TTL 建立索引的时间差别越少，甚至 TAIL 花的时间略超过 TTL 的。像 Berlin 这组数据的线路数很多(线路数多是因为同一线路下存在多条旅程不满足<关系，被拆分成多条线路处理)，平均扫描一次旅程所更新的站点数也不足 3 个(39.9K/15.7K)；同时 TAIL 需要扫描路过各站点的各条线路，Berlin 的站点数和线路数都很大，所以计算量较大。相比之下，TTL 虽然用 O(log n)的复杂度取站点出来扩展，但能保证取出来的站点的最早到达时刻是正确的，省去了反复扫描线路的计算。

5.2 换乘次数

对每组数据，实验随机生成 100,000 个查询。每个查询包含起始站点、目标站点和出发时刻。实验分别对 TAIL 和 TTL 执行这 100,000 个查询求 EAP，然后重构每条路径，得到路径的换乘次数。部分查询返回无解，仅统计有解的路径的平均换乘次数，如表 6 所示。TAIL 索引生成的路径的平均换乘次数都比 TTL 的少，其中 Denver 和 Berlin 两组数据的对比尤为明显。原因如第 4.2 节中所分析，TAIL 在生成索引的时候按旅程扫描路径，保持了旅程的完整性。而 TTL 索引基于图结构，图结构并没能有效地表达出旅程完整性，因此 TTL 生成的路径换乘次数偏多。

表 6　各数据集的路径平均换乘次数

数据集	TAIL	TTL
Austin	5.37	8.27
Budapest	9.52	20.94
Houston	5.09	7.4
Denver	6.86	27.57
Saint Petersburg	8.56	13.93
Rome	7.68	10.47
Toronto	6.26	10.19
Berlin	7.04	26.83

6　相关工作

近年关于公交网络下的路径规划问题有大量的研究工作，这些工作大多数用图结构表示公交网络。Pyrga 等人提出了两种图模型：时间扩展模型和时间依赖模型[7]。时间扩展模型用一个顶点表示“到达(或离开)站点”事件，而时间依赖模型用一个顶点表示一个站点。实验验证，这两种模型中，时间依赖模型生成的图的规模较小，而且用 Time-dependent Dijkstra 算法求解更高效。Daniel 等人提出了一种共享优先队列的方法[8]，用多线程并行计算的方式求解给定出发时间范围的 EAP。Julian 等人提出的 CSA 算法[9]并不采用 Dijkstra 算法。CSA 利用了表示公交网络的图是有向无环图的特性，将(离开站点 A，到达站点 B)的事件按时间排序得到一个事件数组，通过一次扫描事件数组求得 EAP，避免了 Dijkstra 算法的昂贵的堆操作。Wu 等人提出了类似 CSA 的方法[10]，并提出了一种用多线程并行计算求解的方法。上述的方法都是没有建立索引的方法，在较大规模的公交网络图中查询速度较慢。

另一类算法是预先对图做预处理，计算部分路径信息生成索引，然后在索引上做查询。学者们将路网中的求解最短路径的索引思想引入公交网络中，例如有 ALT[11]、SHARC[12]、CHT[13]、Public Transit Labelling[14]等等。但实验表明，这些方法虽然在路网上有效，在公交网络中的加速效果却比不上在路网下的。此外，学者还提出了结合了 CSA[9]思想的 ACSA 算法[15]。Bast 等人提出一种 T. Patterns 方法，该方法预先计算出路径的换乘模式(Transfer pattern)[16]，然后根据换乘模式查找 EAP，但 T.Patterns 并不能保证精确解。TTL[1]是近期提出的一种索引，文[1]中的实验表明 TTL 比 CHT、CSA 等方法更有效。

上述的方法没有考虑到路径的换乘次数，但是在路径规划中，换乘次数是个重要的问题。符等人提出了一种基于节点可达度的方法，能得到最小换乘的多条有效路径[17]。闫等人考虑换乘因素和步行因素的影响，在 Dijkstra 算法的基础上引入迭代惩罚函数搜索路径[18]。Gerth 等人提出了一种多层 Dijkstra 算法，能求出换乘次数为 n 的最早到达路径[19]。Geroge 等人提出了一种求解换乘次数限制的近似算法[20]。Delling 等人提出的 RAPTOR [4]可以在用多线程并行计算，能快速地找到换乘次数为 n 的最早到达路径。但这些算法是在原图上做查询的，速度远比不上带索引的算法。关于公交网络中路径规划问题更详细的讨论可参考 Bast 等人写的综述文章[21]。

7　结束语

本文提出了一种基于旅程扫描和 2-Hub-Labelling 思想的 TAIL 索引。TAIL 与 TTL 索引有相似的结构，因此具有相仿的索引大小与查询时间。与 TTL 相比，TAIL 能有效地减少生成路径的换乘次数。现实公交网络中一般线路数是有限的，当线路下包含的满足<关系的旅程数越多时，TAIL 建立索引的时间比 TTL 的越快。尽管 TAIL 索引能减少生成路径的换乘次数，

但不能保证所得路径必定是换乘次数最少的。下一步工作将研究受换乘次数限制的最快路径的高效索引。

参考文献：

[1] Wand S, Lin Wenqing, Yang Yi, *et al.* Efficient route planning on public transportation networks: a labelling approach [C]// Proc of ACM SIGMOD International Conference on Management of Data. 2015: 967-982.

[2] Abraham I, Delling D, Goldberg A V, *et al.* Hierarchical hub labelings for shortest paths [C]// Proc of European Symposium on Algorithms. Berlin: Springer, 2012: 24-35.

[3] Cooke K L, Halsey E. The shortest route through a network with time-dependent intermodal transit times [J]. Journal of Mathematical Analysis and Applications, 1966, 14 (3): 493-498.

[4] Delling D, Pajor T, Werneck R. F. Round-based public transit routing [C]// Proc of the 14th Meeting on Algorithm Engineering and Experiments Society for Industrial and Applied Mathematics. 2012: 130-140.

[5] Google. GTFS feeds [EB/OL]. http://transitfeeds. com.

[6] Wang Sibo, Lin Wenqing, Yang Yi, *et al.* TTL source code [EB/OL]. https://sites. google. com/site/timelabelling/.

[7] Pyrga E, Schulz F, Wagner D, *et al.* Efficient models for timetable information in public transportation systems [J]. Journal of Experimental Algorithmics, 2008, 12 (1): 2-4.

[8] Delling D, Katz B, Pajor T. Parallel computation of best connections in public transportation networks [J]. ACM Journal of Experimental Algorithmics, 2012, 17 (4): 1–26.

[9] Dibbelt J, Pajor T, Strasser B, *et al.* Intriguingly simple and fast transit routing [C]// Proc of the 12th International Symposium on Experimental Algorithms. Berlin: Springer, 2013: 43–54.

[10] Wu Huanhuan, Cheng J, Huang Silu, *et al.* Path problems in temporal graphs [J]. Proceedings of the VLDB Endowment, 2014, 7 (9): 721-732.

[11] Nannicini G, Delling D, Liberti L, *et al.* Bidirectional A* search on time-dependent road networks [J]. Networks, 2012, 59 (2): 240–251.

[12] Delling D. Time-dependent SHARC-routing [J]. Algorithmica, 2011, 60 (1): 60–94.

[13] Geisberger R, Sanders P, Schultes D, *et al.* Exact routing in large road networks using contraction hierarchies [J]. Transportation Science, 2012, 46 (3): 388–404.

[14] Delling D, Dibbelt J, Pajor T, *et al.* Public transit labeling [M]// Experimental Algorithms. [S. l.] : Springer International Publishing, 2015: 273-285.

[15] Strasser B, Wagner D. Connection scan accelerated [C]// Proc of the 16th Meeting on Algorithm Engineering and Experiments. 2014: 125-137.

[16] Bast H, Carlsson E, Eigenwilling A, *et al.* Fast routing in very large public transportation networks using transfer patterns [C]// Proc of European Symposium on Algorithms. 2010: 290-301, .

[17] 符光梅，王红. 基于节点可达度的公交多路径搜索算法 [J]. 计算机应用研究, 2012, 29 (12): 4492-4494. (Fu Guangmei, Wang Hong. Multi-path search algorithm in public transportation based on node accessibility [J]. Application Research of Computers, 2012, 29 (12): 4492-4494.)

[18] 闫小勇，尚艳亮. 基于二部图模型的公交网络路径搜索算法 [J]. 计算机工程与应用, 2010, 46 (5): 246-248. (Yan Xiaoyong, Shang Yanliang. Path-finding algorithm of public transport networks based on biparticle graphy model [J]. Computer Engineering and Applications, 2010, 46 (5): 246-248)

[19] Brodal G. and Jacob R. Time-dependent networks as models to achieve fast exact time-table queries [C]// Proc of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways. 2004: 3–15.

[20] Tsaggouris G, Zaroliagis C. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications [C]// Proc of the 17th International Symposium on Algorithms and Computation. Berlin: Springer, 2006: 389–398.

[21] Bast H, Delling D, Goldberg A, *et al.* Route planning in transportation networks [C]// Algorithm Engineering. [S. l.] : Springer, 2016: 19-80.